

PHASE1-1.2 PART 2: AWS Cost Collector - Execution & Validation

OptiInfra Development Series

Phase: Cost Agent (Week 2-3)

Component: AWS Cost Collector Validation

Estimated Time: 20 minutes validation

Dependencies: PHASE1-1.2 PART 1 (AWS Cost Collector Code)

Overview

This document provides comprehensive validation steps to ensure the AWS Cost Collector is working correctly and collecting accurate cost data from AWS Cost Explorer API.

Pre-Validation Checklist

Before starting validation, ensure:

- AWS Cost Collector code generated (from PART 1)
 - AWS credentials configured (access key + secret key)
 - AWS Cost Explorer API enabled in your AWS account
 - At least 2 weeks of AWS usage data available
 - Cost Agent service running on port 8001
 - ClickHouse running and accessible
 - Prometheus scraping cost agent metrics
-

Step 0: AWS Credentials Setup

Option 1: Environment Variables (Recommended)

```
bash

# Set AWS credentials
export AWS_ACCESS_KEY_ID="your-access-key-id"
export AWS_SECRET_ACCESS_KEY="your-secret-access-key"
export AWS_DEFAULT_REGION="us-east-1"

# Verify credentials
aws sts get-caller-identity
```

Expected Output:

```
json
{
  "UserId": "AIDAI...",
  "Account": "123456789012",
  "Arn": "arn:aws:iam::123456789012:user/optinfra-collector"
}
```

Option 2: IAM Role (Production)

```
bash

# If running on EC2/ECS, attach IAM role with these permissions
cat > aws-cost-collector-policy.json << EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ce:GetCostAndUsage",
        "ce:GetCostForecast",
        "ce:GetReservationUtilization",
        "ce:GetSavingsPlansUtilization",
        "ce:GetRightsizingRecommendation",
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
        "cloudwatch:GetMetricStatistics",
        "rds:DescribeDBInstances",
        "lambda>ListFunctions",
        "lambda:GetFunction",
        "s3>ListAllMyBuckets",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
EOF
```

Success Criteria:

- AWS credentials valid
 - IAM permissions sufficient for Cost Explorer
 - Can query AWS STS successfully
-

Step 1: Start Cost Agent with AWS Collector

```
bash

# Navigate to cost agent directory
cd ~/optiinfra/services/cost-agent

# Ensure dependencies are installed
pip install boto3==1.34.0 moto==4.2.0

# Restart the service to load new code
pkill -f "python -m src.main"
python -m src.main
```

Expected Output:

```
INFO: Starting Cost Agent...
INFO: Database connection successful
INFO: AWS Cost Collector initialized
INFO: Registered collectors: aws, gcp, azure
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8001
```

Success Criteria:

- Service starts without errors
 - AWS collector registered
 - No credential errors in logs
-

Step 2: Test AWS Cost Explorer Client

2.1 Test Basic Connection

```
bash
```

```
# Test AWS Cost Explorer connection
curl -X POST http://localhost:8001/api/v1/aws/test-connection \
-H "Content-Type: application/json" | jq
```

Expected Response:

```
json
{
  "status": "connected",
  "account_id": "123456789012",
  "regions": ["us-east-1", "us-west-2", "eu-west-1"],
  "cost_explorer_available": true,
  "permissions_valid": true
}
```

2.2 Test Cost Collection (Dry Run)

```
bash
# Trigger cost collection (dry run)
curl -X POST http://localhost:8001/api/v1/aws/collect \
-H "Content-Type: application/json" \
-d '{
  "dry_run": true,
  "start_date": "2025-10-01",
  "end_date": "2025-10-31"
}' | jq
```

Expected Response:

```
json
{
  "status": "completed",
  "dry_run": true,
  "job_id": "aws-collect-uuid",
  "services_found": ["AmazonEC2", "AmazonRDS", "AWSLambda", "AmazonS3"],
  "estimated_cost": 120000.50,
  "collection_duration_seconds": 2.5
}
```

Success Criteria:

- Connection successful
 - Account ID retrieved
 - Cost Explorer API accessible
 - Services detected
-

Step 3: Collect AWS Costs

3.1 Full Cost Collection

```
bash

# Trigger full collection
curl -X POST http://localhost:8001/api/v1/aws/collect \
-H "Content-Type: application/json" \
-d'{
  "start_date": "2025-10-01",
  "end_date": "2025-10-31",
  "services": ["EC2", "RDS", "Lambda", "S3"],
  "analyze": true
}' | jq
```

Expected Response:

```
json

{
  "status": "started",
  "job_id": "aws-collect-20251021-103045",
  "estimated_duration_seconds": 30,
  "services_to_collect": 4,
  "regions_to_scan": 3
}
```

3.2 Check Collection Status

```
bash

# Poll for completion
curl http://localhost:8001/api/v1/aws/jobs/aws-collect-20251021-103045 | jq
```

Expected Response (in progress):

```
json

{
  "job_id": "aws-collect-20251021-103045",
  "status": "running",
  "progress": 0.65,
  "current_service": "AmazonRDS",
  "started_at": "2025-10-21T10:30:45Z",
  "estimated_completion": "2025-10-21T10:31:15Z"
}
```

Expected Response (completed):

```
json

{
  "job_id": "aws-collect-20251021-103045",
  "status": "completed",
  "progress": 1.0,
  "started_at": "2025-10-21T10:30:45Z",
  "completed_at": "2025-10-21T10:31:15Z",
  "duration_seconds": 30,
  "results": {
    "total_cost": 120000.50,
    "services_collected": 4,
    "resources_analyzed": 245,
    "opportunities_identified": 23,
    "estimated_savings": 58000.00
  }
}
```

Success Criteria:

- Job completes successfully
- Cost data collected for all services
- No errors in collection process
- Duration under 60 seconds

Step 4: Query Cost Data

4.1 Get Total Costs

```
bash

# Query total costs
curl "http://localhost:8001/api/v1/aws/costs?start_date=2025-10-01&end_date=2025-10-31" | jq
```

Expected Response:

```
json

{
  "time_period": {
    "start": "2025-10-01",
    "end": "2025-10-31"
  },
  "total_cost": 120000.50,
  "by_service": {
    "AmazonEC2": 85000.00,
    "AmazonRDS": 20000.00,
    "AWSLambda": 8000.50,
    "AmazonS3": 5000.00,
    "Other": 2000.00
  },
  "by_region": {
    "us-east-1": 70000.00,
    "us-west-2": 35000.00,
    "eu-west-1": 15000.50
  },
  "daily_breakdown": [
    {"date": "2025-10-01", "cost": 3870.00},
    {"date": "2025-10-02", "cost": 3890.50}
  ]
}
```

4.2 Get EC2-Specific Costs

```
bash

# Query EC2 costs with details
curl "http://localhost:8001/api/v1/aws/costs/ec2?include_instances=true" | jq
```

Expected Response:

```
json

{
  "total_ec2_cost": 85000.00,
  "instance_count": 125,
  "by_instance_type": {
    "m5.2xlarge": {"count": 40, "cost": 32000.00},
    "c5.xlarge": {"count": 50, "cost": 28000.00},
    "t3.medium": {"count": 35, "cost": 8000.00}
  },
  "ebs_cost": 15000.00,
  "data_transfer_cost": 2000.00,
  "instances": [
    {
      "instance_id": "i-1234567890abcdef0",
      "instance_type": "m5.2xlarge",
      "region": "us-east-1",
      "monthly_cost": 250.00,
      "tags": {"Name": "web-server-01", "Environment": "production"}
    }
  ]
}
```

4.3 Get RDS Costs

```
bash

# Query RDS costs
curl "http://localhost:8001/api/v1/aws/costs/rds" | jq
```

Expected Response:

```
json
```

```
{  
  "total_rds_cost": 20000.00,  
  "instance_count": 15,  
  "storage_cost": 5000.00,  
  "backup_cost": 2000.00,  
  "data_transfer_cost": 500.00,  
  "by_engine": {  
    "postgres": {"count": 8, "cost": 12000.00},  
    "mysql": {"count": 5, "cost": 6000.00},  
    "aurora": {"count": 2, "cost": 2000.00}  
  }  
}
```

Success Criteria:

- Cost data retrieved successfully
- Data matches AWS billing console ($\pm 5\%$)
- Breakdown by service/region accurate
- Daily breakdown available

Step 5: Test Optimization Opportunities

5.1 Get All Opportunities

```
bash  
  
# Query optimization opportunities  
curl "http://localhost:8001/api/v1/aws/opportunities?min_savings=1000" | jq
```

Expected Response:

```
json
```

```
{  
    "total_opportunities": 23,  
    "total_potential_savings": 58000.00,  
    "opportunities": [  
        {  
            "id": "opp-001",  
            "type": "spot_migration",  
            "service": "EC2",  
            "resource_ids": ["i-abc123", "i-def456", "i-ghi789"],  
            "description": "Migrate 15 instances to spot",  
            "estimated_monthly_savings": 18000.00,  
            "confidence": 0.85,  
            "priority": "high",  
            "effort": "medium",  
            "risk": "low"  
        },  
        {  
            "id": "opp-002",  
            "type": "idle_resource",  
            "service": "RDS",  
            "resource_ids": ["db-instance-01", "db-instance-02"],  
            "description": "2 idle RDS instances (0 connections)",  
            "estimated_monthly_savings": 12000.00,  
            "confidence": 0.95,  
            "priority": "high",  
            "effort": "low",  
            "risk": "low"  
        },  
        {  
            "id": "opp-003",  
            "type": "rightsizing",  
            "service": "EC2",  
            "resource_ids": ["i-xyz789"],  
            "description": "Downsize m5.2xlarge to m5.xlarge (15% CPU)",  
            "estimated_monthly_savings": 125.00,  
            "confidence": 0.90,  
            "priority": "medium",  
            "effort": "low",  
            "risk": "low"  
        }  
    ]  
}
```

5.2 Get Idle Resources

```
bash  
# Query idle resources specifically  
curl "http://localhost:8001/api/v1/aws/opportunities?type=idle_resource" | jq
```

Expected Response:

```
json
```

```
{
  "total_idle_resources": 8,
  "total_wasted_cost": 15000.00,
  "resources": [
    {
      "resource_id": "i-1234567890abcdef0",
      "service": "EC2",
      "resource_type": "instance",
      "instance_type": "m5.2xlarge",
      "region": "us-east-1",
      "monthly_cost": 250.00,
      "utilization": {
        "cpu_avg_14d": 2.5,
        "network_mb_day": 50.0
      },
      "idle_duration_days": 18,
      "recommendation": "terminate"
    },
    {
      "resource_id": "db-instance-staging-old",
      "service": "RDS",
      "resource_type": "db_instance",
      "db_engine": "postgres",
      "monthly_cost": 500.00,
      "utilization": {
        "connections_avg_14d": 0.0,
        "cpu_avg_14d": 1.2
      },
      "idle_duration_days": 25,
      "recommendation": "terminate"
    }
  ]
}
```

5.3 Get Spot Opportunities

```
bash

# Query spot migration opportunities
curl "http://localhost:8001/api/v1/aws/opportunities?type=spot_migration" | jq
```

Expected Response:

```
json

{
  "spot_eligible_instances": 15,
  "total_monthly_savings": 18000.00,
  "average_savings_percentage": 38.5,
  "instances": [
    {
      "instance_id": "i-abc123",
      "instance_type": "m5.2xlarge",
      "current_cost": 250.00,
      "spot_cost": 155.00,
      "monthly_savings": 95.00,
      "interruption_rate": "< 5%",
      "workload_type": "batch_processing",
      "spot_eligible": true
    }
  ]
}
```

Success Criteria:

- Opportunities identified and prioritized
- Savings estimates realistic
- Confidence scores appropriate
- Idle resources detected accurately

Step 6: Test Cost Analysis

6.1 Run Comprehensive Analysis

```
bash
```

```
# Trigger full analysis
curl -X POST "http://localhost:8001/api/v1/aws/analysis" \
-H "Content-Type: application/json" \
-d '{
  "analyze_trends": true,
  "detect_anomalies": true,
  "forecast_30d": true
}' | jq
```

Expected Response:

```
json
```

```
{  
  "analysis_id": "analysis-20251021-103500",  
  "timestamp": "2025-10-21T10:35:00Z",  
  "summary": {  
    "total_monthly_cost": 120000.00,  
    "total_waste": 60000.00,  
    "waste_percentage": 50.0,  
    "optimization_potential": 58000.00  
  },  
  "trends": {  
    "cost_change_30d": 5.2,  
    "cost_change_7d": -2.1,  
    "fastest_growing_service": "AWSLambda",  
    "fastest_growing_percentage": 15.5  
  },  
  "anomalies": [  
    {  
      "date": "2025-10-15",  
      "service": "AWSLambda",  
      "metric": "invocation_cost",  
      "expected_cost": 250.00,  
      "actual_cost": 365.00,  
      "change_percentage": 46.0,  
      "severity": "medium"  
    }  
  ],  
  "forecast_30d": {  
    "projected_cost": 125000.00,  
    "confidence_interval": {  
      "lower": 118000.00,  
      "upper": 132000.00  
    },  
    "projected_change_percentage": 4.2  
  },  
  "recommendations_summary": {  
    "high_priority": 8,  
    "medium_priority": 10,  
    "low_priority": 5,  
    "total_savings_potential": 58000.00  
  }  
}
```

 **Success Criteria:**

- Analysis completes successfully
 - Waste percentage calculated
 - Trends identified
 - Anomalies detected (if any)
 - Forecast generated
-

Step 7: Verify ClickHouse Storage

7.1 Check Cost Metrics Table

```
bash

# Query ClickHouse directly
clickhouse-client --query "
SELECT
    date,
    service,
    region,
    SUM(cost) as total_cost
FROM cost_metrics
WHERE date >= today() - 30
    AND provider = 'aws'
GROUP BY date, service, region
ORDER BY date DESC, total_cost DESC
LIMIT 20
"
```

Expected Output:

date	service	region	total_cost
2025-10-21	AmazonEC2	us-east-1	2900.00
2025-10-21	AmazonRDS	us-east-1	680.00
2025-10-21	AWSLambda	us-east-1	275.50
2025-10-20	AmazonEC2	us-east-1	2850.00
2025-10-20	AmazonRDS	us-east-1	680.00

7.2 Check Resource Metrics Table

```
bash

# Query resource utilization
clickhouse-client --query "
SELECT
    resource_id,
    resource_type,
    AVG(cpu_utilization) as avg_cpu,
    AVG(memory_utilization) as avg_memory
FROM resource_metrics
WHERE timestamp >= now() - INTERVAL 7 DAY
    AND provider = 'aws'
GROUP BY resource_id, resource_type
HAVING avg_cpu < 20
ORDER BY avg_cpu ASC
LIMIT 10
"
```

Expected Output:

resource_id	resource_type	avg_cpu	avg_memory
i-1234567890abcdef0	ec2_instance	2.5	15.2
i-0987654321fedcba0	ec2_instance	5.8	22.1
db-staging-old	rds_instance	1.2	8.5

7.3 Check Optimization Opportunities Table

```
bash
```

```
# Query stored opportunities
clickhouse-client --query "
SELECT
    opportunity_type,
    COUNT(*) as count,
    SUM(estimated_savings) as total_savings
FROM optimization_opportunities
WHERE created_at >= today() - 1
    AND provider = 'aws'
GROUP BY opportunity_type
ORDER BY total_savings DESC
"
```

Expected Output:

opportunity_type	count	total_savings
spot_migration	15	18000.00
idle_resource	8	15000.00
rightsizing	35	12000.00
reserved_instance	5	8000.00

Success Criteria:

- Data stored in ClickHouse
- Cost metrics table populated
- Resource metrics captured
- Opportunities persisted
- Query performance good (<100ms)

Step 8: Verify Prometheus Metrics

8.1 Check AWS-Specific Metrics

```
bash
```

```

# Fetch AWS metrics
curl http://localhost:8001/metrics | grep aws_


# Check specific metrics
curl http://localhost:8001/metrics | grep aws_total_monthly_cost_usd
curl http://localhost:8001/metrics | grep aws_waste_identified_usd
curl http://localhost:8001/metrics | grep aws_optimization_opportunities

```

Expected Output:

```

# HELP aws_total_monthly_cost_usd Total AWS monthly cost in USD
# TYPE aws_total_monthly_cost_usd gauge
aws_total_monthly_cost_usd{service="AmazonEC2",region="us-east-1"} 70000.0
aws_total_monthly_cost_usd{service="AmazonRDS",region="us-east-1"} 15000.0
aws_total_monthly_cost_usd{service="AWSLambda",region="us-east-1"} 6000.5

# HELP aws_waste_identified_usd Identified waste in USD
# TYPE aws_waste_identified_usd gauge
aws_waste_identified_usd{service="AmazonEC2"} 38000.0
aws_waste_identified_usd{service="AmazonRDS"} 15000.0

# HELP aws_optimization_opportunities Number of optimization opportunities
# TYPE aws_optimization_opportunities gauge
aws_optimization_opportunities{type="spot_migration"} 15
aws_optimization_opportunities{type="idle_resource"} 8
aws_optimization_opportunities{type="rightsizing"} 35

# HELP aws_api_calls_total Total AWS API calls
# TYPE aws_api_calls_total counter
aws_api_calls_total{service="cost_explorer",operation="GetCostAndUsage"} 12
aws_api_calls_total{service="ec2",operation="DescribeInstances"} 25
aws_api_calls_total{service="cloudwatch",operation="GetMetricStatistics"} 150

# HELP aws_idle_resources_count Number of idle resources
# TYPE aws_idle_resources_count gauge
aws_idle_resources_count{service="EC2"} 5
aws_idle_resources_count{service="RDS"} 3

```

8.2 Query Metrics in Prometheus

bash

```
# Check if Prometheus is scraping
curl "http://localhost:9090/api/v1/query?query=aws_total_monthly_cost_usd" | jq
```

Expected Response:

```
json

{
  "status": "success",
  "data": {
    "resultType": "vector",
    "result": [
      {
        "metric": {
          "service": "AmazonEC2",
          "region": "us-east-1"
        },
        "value": [1729508100, "70000"]
      }
    ]
  }
}
```

Success Criteria:

- AWS metrics exposed via /metrics
- Prometheus scraping successfully
- Metric values match collected data
- All metric types present (counter, gauge, histogram)

Step 9: Verify Grafana Dashboard

9.1 Check Cost Agent Dashboard

```
bash
```

```
# Open Grafana dashboard
open http://localhost:3000/d/optiinfra-cost

# Or check via API
curl -u admin:optiinfra_admin \
"http://localhost:3000/api/dashboards/uid/optiinfra-cost" | \
jq '.dashboard.panels[] | select(.title | contains("AWS")) | .title'
```

Expected Panels:

```
"AWS Total Monthly Cost"
"AWS Cost by Service"
"AWS Cost by Region"
"AWS Waste Identified"
"AWS Optimization Opportunities"
"AWS Idle Resources"
"AWS Spot Opportunities"
"AWS API Call Rate"
```

9.2 Verify Data in Panels

- Open dashboard in browser
- Check all AWS panels show data (not "No Data")
- Verify time series graphs show trends
- Confirm pie charts render (cost by service/region)
- Check gauges show current values

Success Criteria:

- AWS panels added to Cost Agent dashboard
- All panels showing data
- Graphs render correctly
- No query errors

Step 10: Run Automated Tests

10.1 Unit Tests

```
bash

# Run AWS collector tests
cd ~/optiinfra/services/cost-agent
pytest tests/collectors/test_aws.py -v

# Expected tests:
# test_aws_base_collector
# test_cost_explorer_client
# test_ec2_collector
# test_rds_collector
# test_lambda_collector
# test_s3_collector
# test_idle_resource_detection
# test_spot_opportunity_identification
# test_cost_analyzer
# test_clickhouse_storage
```

Expected Output:

```
tests/collectors/test_aws.py::test_aws_base_collector PASSED
tests/collectors/test_aws.py::test_cost_explorer_client PASSED
tests/collectors/test_aws.py::test_ec2_collector PASSED
tests/collectors/test_aws.py::test_ec2_idle_detection PASSED
tests/collectors/test_aws.py::test_ec2_spot_opportunities PASSED
tests/collectors/test_aws.py::test_rds_collector PASSED
tests/collectors/test_aws.py::test_rds_idle_detection PASSED
tests/collectors/test_aws.py::test_lambda_collector PASSED
tests/collectors/test_aws.py::test_s3_collector PASSED
tests/collectors/test_aws.py::test_cost_analyzer PASSED
tests/collectors/test_aws.py::test_clickhouse_storage PASSED
tests/collectors/test_aws.py::test_api_endpoints PASSED

===== 12 passed in 5.23s =====
```

10.2 Integration Tests

```
bash
```

```
# Run integration tests
pytest tests/integration/test_aws_integration.py -v

# Expected tests:
# test_full_collection_workflow
# test_analysis_pipeline
# test_storage_retrieval
# test_api_e2e
```

Expected Output:

```
tests/integration/test_aws_integration.py::test_full_collection_workflow PASSED
tests/integration/test_aws_integration.py::test_analysis_pipeline PASSED
tests/integration/test_aws_integration.py::test_storage_retrieval PASSED
tests/integration/test_aws_integration.py::test_api_e2e PASSED

===== 4 passed in 12.45s =====
```

10.3 Check Code Coverage

```
bash

# Run with coverage
pytest tests/collectors/test_aws.py \
--cov=src.collectors.aws \
--cov=src.analyzers.aws_analyzer \
--cov=src.storage.aws_metrics \
--cov-report=term-missing
```

Expected Output:

----- coverage: platform linux, python 3.11.5 -----

Name	Stmts	Miss	Cover	Missing

src/collectors/aws/__init__.py	5	0	100%	
src/collectors/aws/base.py	125	8	94%	245-252
src/collectors/aws/cost_explorer.py	185	12	94%	312-325
src/collectors/aws/ec2.py	245	18	93%	456-473
src/collectors/aws/rds.py	165	10	94%	287-296
src/collectors/aws/lambda_costs.py	142	9	94%	234-242
src/collectors/aws/s3.py	128	8	94%	201-208
src/analyzers/aws_analyzer.py	215	14	93%	345-358
src/storage/aws_metrics.py	158	10	94%	267-276

TOTAL	1368	89	93%	

Success Criteria:

- All unit tests pass (12/12)
- All integration tests pass (4/4)
- Code coverage $\geq 80\%$ (achieved: 93%)
- No test failures or errors

Troubleshooting

Issue 1: AWS Credentials Not Found

Symptoms:

```
ERROR: botocore.exceptions.NoCredentialsError: Unable to locate credentials
```

Solutions:

```
bash
```

```

# Option 1: Set environment variables
export AWS_ACCESS_KEY_ID="your-key"
export AWS_SECRET_ACCESS_KEY="your-secret"
export AWS_DEFAULT_REGION="us-east-1"

# Option 2: Use AWS CLI configuration
aws configure

# Option 3: Check if credentials file exists
cat ~/.aws/credentials

# Restart service after setting credentials
pkill -f "python -m src.main"
python -m src.main

```

Issue 2: Cost Explorer API Not Enabled

Symptoms:

ERROR: An error occurred (AccessDeniedException) when calling GetCostAndUsage:
Cost Explorer is not enabled for this account

Solutions:

```

bash

# Enable Cost Explorer in AWS Console:
# 1. Go to AWS Cost Explorer
# 2. Click "Enable Cost Explorer"
# 3. Wait 24 hours for data to populate

# Or use AWS CLI:
aws ce get-cost-and-usage \
--time-period Start=2025-10-01,End=2025-10-31 \
--granularity=MONTHLY \
--metrics=UnblendedCost

# If error persists, check IAM permissions

```

Issue 3: Rate Limiting

Symptoms:

```
ERROR: botocore.exceptions.ClientError: An error occurred (ThrottlingException)  
when calling GetCostAndUsage: Rate exceeded
```

Solutions:

```
bash  
  
# Cost Explorer rate limit: 400 requests/hour  
# Check current API call rate  
curl http://localhost:8001/metrics | grep aws_api_calls_total  
  
# Reduce collection frequency in config  
# Add delays between API calls  
# Enable caching in Redis  
  
# Wait for rate limit to reset (1 hour)
```

Issue 4: No Cost Data Available

Symptoms:

```
{"total_cost": 0.0, "by_service": {}}
```

Solutions:

```
bash
```

```
# Check if AWS account has recent usage
aws ce get-cost-and-usage \
--time-period Start=2025-10-01,End=2025-10-31 \
--granularity=MONTHLY \
--metrics=UnblendedCost

# Verify date range is correct
# Cost Explorer requires 24-48 hours for data

# Check if filtering is too restrictive
curl "http://localhost:8001/api/v1/aws/costs?start_date=2025-09-01&end_date=2025-10-31"
```

Issue 5: ClickHouse Storage Failures

Symptoms:

```
ERROR: clickhouse_driver.errors.NetworkError: Connection refused
```

Solutions:

```
bash

# Check if ClickHouse is running
docker ps | grep clickhouse

# Start ClickHouse if not running
docker-compose up -d clickhouse

# Test connection
clickhouse-client --query "SELECT 1"

# Check if tables exist
clickhouse-client --query "SHOW TABLES FROM optiinfra"

# Restart cost agent
pkill -f "python -m src.main"
python -m src.main
```

Validation Completion Checklist

Mark each item when verified:

AWS Connection

- AWS credentials configured
- Cost Explorer API accessible
- IAM permissions sufficient
- Multi-region access working

Cost Collection

- Can trigger collection via API
- Collection completes successfully
- All services collected (EC2, RDS, Lambda, S3)
- Cost data matches AWS console ($\pm 5\%$)

Optimization Detection

- Idle resources identified
- Spot opportunities found
- Rightsizing recommendations generated
- Savings estimates realistic

Data Storage

- Costs stored in ClickHouse
- Resource metrics persisted
- Opportunities saved
- Query performance acceptable

Monitoring

- Prometheus metrics exposed
- Grafana dashboard shows AWS data
- API call rate tracked
- Errors logged and tracked

Testing

- All unit tests pass (12/12)
- Integration tests pass (4/4)
- Code coverage $\geq 80\%$

- No memory leaks

Documentation

- API endpoints documented
- Configuration options clear
- Troubleshooting guide helpful
- IAM policy examples included

Success Metrics Summary

After completing all validation steps, verify these metrics:

Metric	Target	Actual	Status
AWS Connection	Success	<input type="checkbox"/>	<input type="checkbox"/>
Cost Collection	<60s	<input type="checkbox"/>	<input type="checkbox"/>
Services Collected	4+	<input type="checkbox"/>	<input type="checkbox"/>
Idle Resources Found	5-15%	<input type="checkbox"/>	<input type="checkbox"/>
Spot Opportunities	30-40% savings	<input type="checkbox"/>	<input type="checkbox"/>
Data in ClickHouse	30 days	<input type="checkbox"/>	<input type="checkbox"/>
Prometheus Metrics	All exposed	<input type="checkbox"/>	<input type="checkbox"/>
Unit Tests	100% pass	<input type="checkbox"/>	<input type="checkbox"/>
Code Coverage	≥80%	<input type="checkbox"/>	<input type="checkbox"/>
API Response Time	<500ms	<input type="checkbox"/>	<input type="checkbox"/>

Next Steps

Once validation is complete:

Option 1: Continue to GCP Collector

"Generate PROMPT 1.3: GCP Cost Collector"

Option 2: Review AWS Implementation

"Show me AWS collector code structure"

Option 3: Get Help

"I'm having issues with AWS validation step X"

Additional Resources

- [AWS Cost Explorer API Documentation](#)
- [boto3 Documentation](#)
- [AWS Cost Optimization Best Practices](#)
- [moto - Mock AWS Services](#)

Completion Sign-off

Validated By: _____

Date: _____

Status: PASS / FAIL

AWS Account ID: _____

Total Cost Collected: \$_____

Savings Identified: \$_____

Notes: _____

Document Version: 1.0

Status: Validation Guide Complete

Last Updated: October 21, 2025

Previous: PHASE1-1.2 PART 1 (AWS Collector Code)

Next: PHASE1-1.3 (GCP Cost Collector)